Interactive Proofs For Differentially Private Counting

ABSTRACT

Differential Privacy (DP) is often presented as a strong privacy-enhancing technology with broad applicability and advocated as a de-facto standard for releasing aggregate statistics on sensitive data. However, in many embodiments, DP introduces a new attack surface: a malicious entity entrusted with releasing statistics could manipulate the results and use the randomness of DP as a convenient smokescreen to mask its nefariousness. Since revealing the random noise would obviate the purpose of introducing it, the miscreant may have a perfect alibi. To close this loophole, we introduce the idea of *Interactive Proofs For Differential Privacy*, which requires the publishing entity to output a zero knowledge proof that convinces an efficient verifier that the output is both DP and reliable. Such a definition might seem unachievable, as a verifier must validate that DP randomness was generated faithfully without learning anything about the randomness itself. We resolve this paradox by carefully mixing private and public randomness to compute verifiable DP counting queries with theoretical guarantees and show that it is also practical for real-world deployment. We also demonstrate that computational assumptions are necessary by showing a separation between information-theoretic DP and computational DP under our definition of verifiability.

CCS CONCEPTS

• **Security and privacy**;

KEYWORDS

differential privacy, secure multiparty computation, verifiable computation, zero knowledge

ACM Reference Format:

. 2018. Interactive Proofs For Differentially Private Counting. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

We are living in an age of delegation, where the bulk of our digital data is held and processed by others in an opaque fashion. Our interactions are collated by digital applications that continually send our personal information to the "cloud". Servers in the cloud, typically owned by large monolithic organizations, such as Google, AWS or Microsoft, then perform computations on our private data to publish aggregate statistics for social utility. For example, we send our GPS coordinates to services like Strava and Google which, in exchange, use this information to recommend low-traffic cycling routes [52]. Similarly, we let entertainment companies like Netflix, YouTube, TikTok and Hulu know our personal preferences so that they can better recommend content for us to consume [10]. National census bureaus collect personal information to publish aggregate statistics about the population, and consider doing so a moral duty to ensure transparency in the government's policies [16].

However, it is often the case that published aggregate statistics leak information about the activity of individuals. For example, Garfinkel *et al.* and Kasiviswanathan *et al.* describe practical reconstruction attacks that can be used to infer an individual's private data from aggregate population statistics [35, 44]. Boyd *et al.* show that published census data has been used to discriminate against groups in society based on race [16]. Hence the information that is released, and how it is computed, requires careful scrutiny.

In response to these concerns, the privacy and security community have sought to apply various privacy enhancing technologies to protect the privacy of individuals contributing to data releases. Most relevant to this discussion is Differential Privacy (DP) and its generalizations, which require computations to be randomized, in order to offer the (informally stated) promise that users will not be adversely affected by allowing their data to be used. Typically, this is achieved by adding carefully calibrated random noise to the output, at the expense of reducing the accuracy of the computation. Differential privacy is most commonly studied in the *trusted curator* model, where a single entity receives all the sensitive data, and is entrusted to execute the algorithm to apply the random noise. Variations that modify the trust and computational model include local privacy [57], shuffle privacy [5, 21], computational differential privacy [50] and multi-party differential privacy [48].

A consistent theme across all existing work is to view DP simply as a privacy-preserving mechanism. In this paper, we shift the focus and view differential privacy through an adversarial lens: *what if the entity responsible for releasing aggregate DP statistics seeks to abuse the protocol and pick noise chosen to distort the statistics, using differential privacy as an attack vector?*

That is, a malicious entity may tamper with the computation in order to publish biased statistics, and claim this reflects the true outcome; any discrepancies may be dismissed as artifacts of random noise. Consider a counting query DP protocol to determine the winner of a plurality election, where the users vote for 1 out of M candidates (say, which topping people prefer on their pizza). A corrupted aggregator might not be interested in any particular user's vote but in biasing the aggregate output of the protocol instead. Thus, if that server has auxiliary information about the preferences of a subset of users, they might tamper with the protocol to exclude those honest voters from the election or tamper with the protocol to bias the results of the election (say, to pineapple) and blame any discrepancies in the result on random noise introduced by DP. Note that some loss in accuracy for privacy is unavoidable. By definition, DP requires the output be perturbed by private randomness. Often, outputting such random statistics creates tensions between publishing entity and the downstream consumer. In 2021, the State Of

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00

https://doi.org/XXXXXXX.XXXXXXX

Alabama filed a lawsuit claiming that the use of DP on census data was illegal [42], citing the inaccuracies introduced by DP. Thus, to ensure public trust in DP, it is critical to verify that any loss in utility can be attributed solely to unavoidable DP randomness.

To that effect, we formally introduce the idea of *Interactive Proofs For Differential Privacy* in both the trusted curator [1] setting and the multi-party setting in presence of active adversaries [2]. Our contributions are as follows:

1. We formally introduce definitions for *Interactive Proofs For Differential Privacy* in both the trusted curator and client-server multiparty setting [6]. Informally, the entity responsible for releasing DP statistics must also output a zero knowledge proof to verify that the statistic was computed correctly with respect to committed client inputs and the private randomness generated faithfully. Such a proof reveals no additional information and still enforces that user privacy is protected via DP but ensures that the curator cannot use DP randomness maliciously.
2. We show concrete instantiations of verifiable DP by computing DP counting queries (histograms) in the trusted curator and client-server multiparty settings. In the trusted curator setting, there is a single aggregating server that sees client data in plaintext and is responsible for outputting a DP histogram and a proof that the DP noise was generated faithfully. In the client-server MPC setting, clients secret share the inputs and send them to $K \geq 2$ servers, who then participate in an MPC protocol to output DP histograms. The protocol itself is secure in that not even the participating servers are able to learn any new information beyond the output nor are they able to tamper with the protocol.
3. We conduct experiments to show that our protocols with formal theoretical guarantees are also practical. Additionally, we describe how our protocol $\Pi_{\mathtt{Bin}}$, for verifiable DP counting, can be combined with existing (non-verifiable) DP-MPC protocols, such as PRIO [25] and Poplar [15], to enforce verifiability.
4. We demonstrate that information-theoretic verifiable DP is impossible. Specifically, if both the prover and verifier are computationally unbounded, then statistical zero knowledge and unconditional soundness cannot hold simultaneously. Thus we could either prevent an all-powerful curator from manipulating DP protocols or an all-powerful verifier from being able to distinguish between neighbouring datasets from the output, but not both. This result is related to an open problem (Open Problem 10.6) of Vadhan [56], which asks *"Is there a computational task solvable by a single curator with computational differential privacy but is impossible to achieve with information-theoretic differential privacy?"*. In Section 5 we relate our result to efforts at resolving this question.

2 PRELIMINARIES

2.1 Notation

We write $x \overset{R}{\longleftarrow} U$ to denote that x was uniformly sampled from a set U. We denote vectors with an arrow on top as in $\vec{x} \in \mathbb{Z}_q^M$, where M represents the number of coordinates in the vector and $\mathbb{Z}_q$ represents a prime order finite field of integers of size q. We write $\vec{a} + \vec{b}$ to mean coordinate-wise vector addition $a + b \mod q$, where a and b correspond to values at the same position of $\vec{a}$ and $\vec{b}$. Similarly, when we write $\vec{a} \times \vec{b}$, we refer to the coordinate-wise Hadamard product between the two vectors.

2.2 Privacy and Security Background

Indistinguishability. We define a computational notion of indistinguishability.

Definition 2.1 (Computational Indistinguishability). Fix security parameter $\kappa \in \mathbb{N}$. Let $\{X_\kappa\}_{\kappa \in N}$ and $\{Y_\kappa\}_{\kappa \in N}$ be probability distributions over $\{0,1\}^{\mathtt{poly}(\kappa)}$. We say that $\{X_\kappa\}_{\kappa \in N}$ and $\{Y_\kappa\}_{\kappa \in N}$ are computationally indistinguishable $\{X_\kappa\}_{\kappa \in N} \overset{c}{\equiv} \{Y_\kappa\}_{\kappa \in N}$ if for all non-uniform PPT turing machines D ("distinguishers"), there exists a negligible function $\mu(\kappa)$ such for every $\kappa \in \mathbb{N}$

$$\left| \Pr[D(X_\kappa) = 1] - \Pr[D(Y_\kappa) = 1] \right| \leq \mu(\kappa) \qquad (1)$$

Commitments. Commitments are used in our schemes to ensure that participants cannot change their response during the protocol.

Definition 2.2 (Commitments). Let $\kappa \in \mathbb{N}$ be the security parameter. A non-interactive commitment scheme consists of a pair of probabilistic polynomial time algorithms ($\mathtt{Setup}$, $\mathtt{Com}$). The setup algorithm $\mathsf{pp} \leftarrow \mathtt{Setup}(1^\kappa)$ generates public parameters pp. Given a message space M_{pp} and randomness space R_{pp}, the commitment algorithm $\mathtt{Com}_{\mathsf{pp}}$ defines a function $\mathsf{M}_{\mathsf{pp}} \times \mathsf{R}_{\mathsf{pp}} \longrightarrow \mathsf{C}_{\mathsf{pp}}$ that maps a message to the commitment space C_{pp} using the random space. For a message $x \in \mathsf{M}_{\mathsf{pp}}$, the algorithm samples $r_x \overset{R}{\longleftarrow} \mathsf{R}_{\mathsf{pp}}$ and computes $c_x = \mathtt{Com}_{\mathsf{pp}}(x, r_x)$. When the context is clear, will drop the subscript and write $\mathtt{Com}_{\mathsf{pp}}$ as $\mathtt{Com}$.

Definition 2.3 (Homomorphic Commitments). A homomorphic commitment scheme is a non-interactive commitment scheme such that M_{pp} and R_{pp} are fields (with $(+, \times)$) and C_{pp} is an abelian group with the $\otimes$ operator, such that for all $x_1, x_2 \in \mathsf{M}_{\mathsf{pp}}$ and $r_1, r_2 \in \mathsf{R}_{\mathsf{pp}}$ we have

$$\mathtt{Com}(x_1, r_1) \otimes \mathtt{Com}(x_2, r_2) = \mathtt{Com}(x_1 + x_2, r_1 + r_2) \qquad (2)$$

Throughout this paper, when we use a commitment scheme, we mean a non-interactive homomorphic commitment scheme with the following properties (stated informally here, but formalized in the Appendix A):

1. **Hiding:** A commitment c_x reveals no information about x and r_x to a computationally bounded adversary (Definition A.2).
2. **Binding:** Given a commitment c_x to x using r_x, there is no efficient algorithm that can find x' and $r_{x'}$ such that $\mathrm{Com}(x', r_{x'}) = c_x = \mathrm{Com}(x, r_x)$ (Definition A.3).

[1] When we say trusted curator, we imply that there is a single server that can view client inputs in plaintext. However, this server could still be corrupted and therefore, it must prove that the final released output was computed as prescribed the DP protocol. Of course in the single server setting we cannot protect client privacy. The focus is on ensuring the output is reliable

[2] By active adversaries, we mean participants that may deviate from protocol specifications arbitrarily. In the MPC setting we can guarantee both privacy of client inputs and reliability of output.

(3) **Zero Knowledge OR Opening:** Given c_x, the committing party can prove to a polynomial time verifier that c_x is a commitment to either 1 or 0 without revealing which one it is. We denote such a proof as $\Pi_{\texttt{OR}}$ and say it securely computes the oracle $O_{\texttt{OR}}$, which returns true if $c_x \in L_{\texttt{Bit}}$

$$L_{\texttt{Bit}} = \{c_x : x \in \{0,1\} \wedge r_x \in \mathbb{Z}_q \wedge c_x = \mathrm{Com}(x, r_x)\} \tag{3}$$

See Appendix B for a concrete construction of the Σ-OR proof using Pedersen Commitment schemes from [27].

In all our experiments and security proofs, we use Pedersen Commitments (PC), though one could replace PC with [7, 30, 58], and still satisfy all the above properties.

Differential Privacy (DP and IND-CDP). We consider two variants of the privacy definition.

Definition 2.4 (Information Theoretic DP [56]). Fix $n \in \mathbb{N}, \epsilon \geq 0$ and $\delta \leq n^{-\omega(1)}$. An algorithm $\mathcal{M} : \mathcal{X}^n \times Q \to \mathcal{Y}$ satisfies (ϵ, δ) differential privacy if for every two neighboring datasets $X \sim X'$ s.t. $||X - X'||_1 = 1$ and for every query $Q \in \mathcal{Q}$ we have for all $T \subseteq \mathcal{Y}$

$$\Pr\big[M_Q(X) \in T\big] \leq e^{\epsilon} \Pr\big[M_Q(X') \in T\big] + \delta \tag{4}$$

A direct corollary of the above definition is that, given $M_Q(X)$ and $M_Q(X')$, with probability $1 - \delta$ even an unbounded Turing Machine (TM) D is unable to distinguish between the outputs up to statistical distance ϵ.

Definition 2.5 (Computational DP [50]). Fix $\kappa \in \mathbb{N}$ and $n \in \mathbb{N}$. Let $\epsilon \geq 0$ and $\delta(\kappa) \leq \kappa^{-\omega(1)}$ be a negligible function, and let $\mathcal{M} = \{\mathcal{M}_\kappa : \mathcal{X}_\kappa^n \to \mathcal{Y}_\kappa\}_{\kappa \in \mathbb{N}}$ be a family of randomised algorithms, where $\mathcal{X}_\kappa$ and $\mathcal{Y}_\kappa$ can be represented by $\texttt{poly}(\kappa)$-bit strings. We say that $\mathcal{M}$ is *computationally ϵ-differentially private* if for every non-uniform PPT TM's ("distinguisher") D, for every query $Q \in \mathcal{Q}$, and every neighbouring dataset $X \sim X' \in \mathcal{X}_\kappa^n, \forall T \subseteq \mathcal{Y}_\kappa$ we have

$$\Pr\Big[D(\mathcal{M}_\kappa(X, Q) \in T) = 1\Big] \leq e^{\epsilon} \cdot \Pr\Big[D(\mathcal{M}_\kappa(X', Q) \in T) = 1\Big] + \delta(\kappa) \tag{5}$$

Definition 2.6. (DP-Error) Let $\mathcal{M} : \mathcal{X} \times \mathcal{Q} \to \mathcal{Y}$ be a (ϵ, δ)-DP mechanism over $\mathcal{Q}$. Assume that the L_1 norm is well-defined on $\mathcal{Y}$. For any $n \in \mathbb{N}$, $X \in \mathcal{X}^n$, we define the expected error of the mechanism $\mathcal{M}$ relative to Q as

$$\texttt{Err}_{\mathcal{M},Q} = \mathbb{E}[\|Q(X) - \mathcal{M}_Q(X)\|_1] \tag{6}$$

where the expectation is taken over internal randomness of $\mathcal{M}$.

When the context is clear, to simplify notation we drop subscripts and refer to equation (6) as just $\texttt{Err}$. It is well known that for negligible δ, the counting query (i.e., DP histograms) has error $\texttt{Err} = O(\frac{1}{\epsilon})$ in the trusted curator model and MPC model [25, 56].

Binomial Mechanism. We use Binomial noise to achieve privacy.

Lemma 2.7 (Binomial Mechanism). *Let $X = (x_1, \ldots, x_n) \in \mathbb{Z}_q^n$ and define counting query $Q(X) = \sum_{i=1}^n x_i$. Fix $n_b > 30$, $0 < \delta \leq o(\frac{1}{n_b})$ and let $Z \sim \texttt{Binomial}(n_b, \frac{1}{2})$. Then $Z + Q(X)$ is (ϵ, δ)-differentially private with $\epsilon = 10\sqrt{\frac{1}{n_b} \ln \frac{2}{\delta}}$.*

It is easy to see that the binomial mechanism incurs constant DP error (i.e., it is independent of n and depends only on ϵ, δ). The proof for Lemma 2.7 can be found in [36].

Algorithm 1 Π_{morra} A protocol for sampling a public coin

Input: $\lambda_1, \ldots, \lambda_K$

Output: $z \overset{R}{\longleftarrow} \{0, 1\}$

(1) Each server $k \in [K]$ is asked to sample $m_k \overset{R}{\longleftarrow} \mathbb{Z}_q$ uniformly at random.

(2) *Commit*: Each server samples $r_{m_k} \overset{R}{\longleftarrow} \mathbb{Z}_q$ and broadcasts $c_k = \mathrm{Com}(m_k, r_k)$ to all other servers. Assume without loss of generality that the servers broadcast their commitments in natural lexicographical order $k \in [K]$.

(3) *Reveal*: Once all servers have received c_k, they now broadcast m_k, r_{m_k} to all servers in the reverse order in which the commitments arrived. It is important that the reverse order is respected as it guarantees that each server's inputs are independent of the inputs of other servers. Once all commitments are revealed, each server verifies that $\mathrm{Com}(m_k, r_k) = c_k$. If this test fails for any k or one of the servers does not respond, the protocol is aborted.

(4) Each server computes $X = (m_1 + \cdots + m_k) \mod q$. We have $X \overset{R}{\longleftarrow} \mathbb{Z}_q$. If $X \leq \lceil \frac{q}{2} \rceil$ then $c_i = 0$. Otherwise $c_i = 1$. Thus we can use this protocol to generate unbiased coins and uniformly random values.

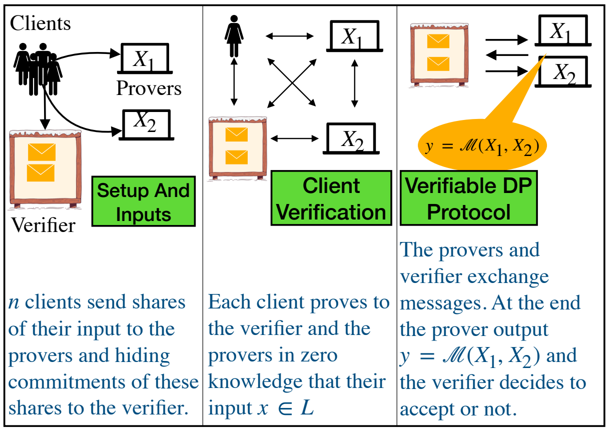
Morra. We will prove zero knowledge (or security for MPC) assuming that the provers and verifiers (or all participants of the MPC, respectively) have access to an oracle that returns a polynomial sized stream of publicly random unbiased bits. In other words, we assume that all parties have access to an oracle functionality $O_{\texttt{morra}}(1^\kappa, \lambda_1, \ldots, \lambda_K) = z$ where $z \overset{R}{\longleftarrow} \{0, 1\}$ and λ_k refers to the empty string for all $k \in [K]$.

In practice, this oracle is replaced by a lightweight MPC protocol such as $\Pi_{\texttt{morra}}$ defined in Algorithm 1, which is a modification of an ancient game called Morra[3], that securely computes $O_{\texttt{morra}}$ in the presence of a dishonest majority of active participants. It is easy to see that as long as one participant is honest and samples its value uniformly at random, the final protocol produces an unbiased coin. Since the commitment is hiding, a corrupt party cannot infer any information about the other parties choice m_k from the published c_{m_k} and by the binding property, a participant cannot change their decision after observing another party's opening. A formal simulator-styled proof can be found in Blum's seminal work for flipping coins over a telephone [12] or any introductory textbook on MPC (under the title weak coin flipping). If we omit the final thresholding step, the above protocol can be used to sample $z \overset{R}{\longleftarrow} Z_q$.

3 SECURITY MODELS FOR VERIFIABLE DP

This section introduces verifiable DP in both the single trusted curator and MPC model. In both settings, the input for the protocol comes from n distinct clients. Informally, the main difference between the two models is that the former has plaintext access to the client data. In contrast, in MPC-DP, the clients secret share (or partition) their inputs and each server receives information

[3] https://en.wikipedia.org/wiki/Morra_(game)



n clients send shares of their input to the provers and hiding commitments of these shares to the verifier.

Each client proves to the verifier and the provers in zero knowledge that their input $x \in L$

The provers and verifier exchange messages. At the end the prover output $y = \mathscr{M}(X_1, X_2)$ and the verifier decides to accept or not.

Figure 1: The figure above describes the three stages of the protocol. Any message sent and received by the verifier is accessible to all clients and provers. In the setup stage, public parameters are generated, and each client $i \in [n]$ sends inputs to the prover and the verifier. In the verification stage, each client interactively exchanges messages with the verifier and the provers to establish their private input $x \in L$ in zero knowledge. If a client fails to do so, they are tagged as dishonest and excluded from the protocol. In the last stage, each prover samples private randomness and then interactively exchanges messages with other provers and the verifier to jointly compute $y = \mathcal{M}_Q(X_1, \dots, X_K)$ for some common knowledge query Q. The verifier validates that the provers' output y was computed as prescribed over the inputs of honest clients only.

theoretically hiding shares (or a partial view) of client inputs. Additionally, instead of a single trusted entity computing $\mathcal{M}$, the servers participate in an MPC protocol Π to securely compute $\mathcal{M}$ without revealing any information other than $\mathcal{M}_Q(x_1, \dots, x_n)$.

For some queries $Q \in \mathcal{Q}$, the protocol requires that the client inputs come from a restricted subset $L \subseteq \mathcal{X}$. For such cases, the clients must send a zero knowledge proof so that we can verify that the inputs come from the specified language without learning any other information about the inputs. Examples of such proofs can be found in the prior literature [15, 18, 20, 25]. In the definitions below and in what follows, we use the terms Pv (prover), server and curator interchangeably, and the terms analyst and Vfr (verifier) to refer to the same entity.

3.1 Verifiable DP

Next, we describe the MPC model and later discuss how it can be specialised to the trusted curator model. Let $\mathcal{M}_Q$ be a DP (or IND-CDP) mechanism as described in Definition 2.4 (or Definition 2.5 respectively) for a query $Q \in \mathcal{Q}$. Let $\kappa \in \mathbb{N}$ denote the security parameter. A verifiable DP mechanism for $\mathcal{M}_Q$ consists of three interactive protocols $(\mathsf{Setup}, \mathsf{Verify}, \Pi)$, between n clients that have private inputs $(x_1, \dots, x_n) \in \mathcal{X}$ and $K{+}1$ "next-message-computing-algorithms" Vfr and $(\mathsf{Pv}_1, \dots, \mathsf{Pv}_K)$. In next-message computing algorithms, party V's message m_i at round i is determined by its input, messages it has received so far from other parties and internal randomness $\vec{r}_V$. Let $\vec{\mathsf{Pv}}$ denote a succinct representation for $(\mathsf{Pv}_1, \dots, \mathsf{Pv}_K)$.

In Setup, all parties jointly generate public parameters and the provers and verifier receives inputs from n clients. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$ denote public parameters. Each prover Pv_k receives on its input tape n secret shares of client inputs $\left([\![x_1]\!]_k, \dots, [\![x_n]\!]_k\right)$, succinctly denoted by $\vec{X_k}$. The verifier receives hiding commitments of the above shares. All messages sent and received by the verifier are accessible to all other parties.

If the query Q restricts client inputs to a subset $L \subseteq \mathcal{X}$ then in Verify phase, the clients interactively exchange messages with the provers and the verifier to prove in zero knowledge that their private input $x \in L$. If clients fail to do so, they are excluded from the protocol.

Once dishonest clients with illegal inputs have been excluded and honest client inputs have been recorded, the clients play no further role in the protocol. The third protocol Π describes a multi-party interactive proof system, where the provers and the verifiers interactively exchange messages for $\mathsf{poly}(\kappa)$ rounds. Let $z \in \{0,1\}^{\mathsf{poly}(\kappa)}$ denote auxiliary input available to the verifier. At the end of the protocol, the provers send $\vec{y} \in \mathcal{Y}$ to the Vfr, who then outputs either 0 or 1, with 1 indicating that the verifier accepts the provers' claim that, the real protocol output is indistinguishable from an ideal computation, i.e. $\vec{y} = \mathcal{M}_Q(X)$. Let $\mathsf{out}\left[\mathsf{Vfr}(\mathsf{pp}, \vec{r}_v, z), \vec{y}, \vec{\mathsf{Pv}}(\mathsf{pp}, \vec{r}_{\vec{\mathsf{Pv}}})\right] \in \{0,1\}$ denote the verifying algorithm's decision. In the definition below, we write $\mathsf{out}(\mathsf{Vfr}, \mathsf{Pv})$ as shorthand for the verifier output.

The trusted curator can be understood as essentially this model with a single prover, i.e., we set $K = 1$. Thus the only functional difference between MPC-DP and trusted curator DP is that in the latter case, the curator sees all the data in plaintext. In MPC, the data may be secret, shared or partitioned across the provers. In both cases, the prover(s) must prove they did not tamper with the protocol to generate an output distinguishable from the ideal computation of $\mathcal{M}$ [4]. Figure 1 summarises the information flow between the parties.

Definition 3.1 (Verifiable DP). An interactive verifiable DP protocol for $\mathcal{M}$ is an interactive message passing protocol Π, such that for $n \in \mathbb{N}$ honest clients, $K \geq 1$ provers denoted by $\vec{\mathsf{Pv}}$ and a single verifier Vfr, there exists negligible functions δ_c and δ_s such that the following hold:

(1) **Completeness:** Let $X = (x_1, \dots, x_n) \in \mathcal{X}^n$ be the legal client inputs that have been split in K shares $(\vec{X_1}, \dots, \vec{X_K})$, where $\vec{X_j}$ denotes the input sent to the j'th prover, then as long as the $\vec{\mathsf{Pv}}$ and Vfr honestly execute Π, then we have

$$\Pr\left[\mathsf{out}(\mathsf{Vfr}, \vec{\mathsf{Pv}}) = 0 : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa) \\ \mathsf{Pv}_j \leftarrow (\vec{X_j}, \vec{r}_{\mathsf{Pv}_j}, \mathsf{pp}) \\ \mathsf{Vfr} \leftarrow (z, \vec{r}_v, \mathsf{pp}) \\ \vec{y} \leftarrow \Pi(\vec{\mathsf{Pv}}, \mathsf{Vfr}, \mathsf{pp}) \end{array}\right] \leq \delta_c.$$

[4] When there is a single server only, the server can see inputs in plaintext. Thus in the Verify phase, the clients only need to prove to the verifier in zero knowledge that the inputs are legal

(2) **Soundness:** Let $X = (x_1, \ldots, x_n) \in \mathcal{X}^n$ be the legal client inputs. For any subset $I \subseteq [K]$, let $\vec{\mathsf{Pv}}^*$ denote the collection of corrupted provers, indexed by I, that deviate from Π, and $\vec{\mathsf{Pv}}$ denote the set of honest provers not indexed by I. For any output $\vec{y} \neq \mathcal{M}(X, Q)$, we have

$$\Pr\left[\mathtt{out}(\mathtt{Vfr}, \vec{\mathsf{Pv}}, \vec{\mathsf{Pv}}^*) = 1 : \begin{array}{c} \mathsf{pp} \leftarrow \mathtt{Setup}(1^\kappa) \\ \mathsf{Pv}_j \leftarrow (\vec{X}_j, \vec{r}_{\mathsf{Pv}_j}, \mathsf{pp}) \\ \mathtt{Vfr} \leftarrow (z, \vec{r}_v, \mathsf{pp}) \\ \vec{y} \leftarrow \Pi(\vec{\mathsf{Pv}}^*, \mathtt{Vfr}, \mathsf{pp}) \end{array}\right] \leq \delta_s.$$

Note that the correctness of the protocol is defined in terms of the actual inputs the clients sent to $\mathcal{M}$ and not the inputs a corrupted set of provers might have used.

(3) **Zero Knowledge**: For any subset $I \subseteq [K]$, let $\vec{\mathsf{Pv}}^*$ denote the collection of corrupted provers, indexed by I, that deviate from Π, and $\vec{\mathsf{Pv}}$ denote the set of honest provers. Let $\mathtt{View}\Big[\Pi\Big((\vec{\mathsf{Pv}}, \vec{\mathsf{Pv}}^*), \mathtt{Vfr}^*\Big)\Big]$ be the joint distribution[5] of messages and output received during the execution of Π in the presence of corrupted parties. There exists a PPT Simulator $\mathsf{Sim}_{(\mathtt{Vfr}^*, I)}$ with black box access to $\mathtt{Vfr}^*$ and $\vec{\mathsf{Pv}}^*$, such that for all $\vec{y} = \mathcal{M}(X, Q)$

$$\mathtt{View}\Big[\Pi\Big((\vec{\mathsf{Pv}}, \vec{\mathsf{Pv}}^*), \mathtt{Vfr}^*\Big)\Big] \equiv \mathsf{Sim}_{(\mathtt{Vfr}^*, I)}(\vec{y}, \vec{r}_v, z, \mathsf{pp})$$

Contrary to soundness, for zero knowledge to hold, the simulated transcript should be indistinguishable from the actual protocol transcript, based on the inputs adversaries used and not the ones the clients sent to a set of corrupted provers.

An interesting point to note is that in verifiable differential privacy, the verifier plays a dual role. An honest verifier ensures that the output is faithfully generated and thus plays an active role in generating the DP noise without ever seeing it in plaintext. On the other hand, a dishonest verifier tries to tamper with the protocol to breach privacy. In non-verifiable DP, the analysts (verifier) only have access to the DP statistic. They have no agency over how the output is generated. Thus the verifier participating in verifiable DP has a greater attack surface than a classical adversary in traditional non-verifiable DP. We elaborate on this in Section 5, when trying to establish separations between statistical DP and computational DP. Additionally, just like in standard MPC, in the presence of a dishonest majority of corrupted participants, we do not treat early exiting by corrupted parties as a breach of security. This is easily detected by the honest parties, and the output is ignored. Verifiable DP, just like interactive zero knowledge proofs [37] comes in 24 different flavours based on the capabilities of the corrupted parties:

(1) **Distinguishability:** Based on the distinguishability properties of the simulator algorithm, the protocol may be perfect, statistical or computationally zero knowledge. The protocol described in Section 4 is computationally zero knowledge.

(2) **Verifier specifications:** Based on whether the verifier is expected to follow the rules of the protocol (semi-honest) or may deviate arbitrarily (active), we get honest-verifier zero knowledge or unrestricted zero knowledge. All our results are zero knowledge.

(3) **Soundness:** Based on the power of the corrupted provers, the proof may be computationally sound (also known as arguments) or statistically sound (secure against unbounded provers). The verifiable DP protocol in Section 4 is computationally sound.

(4) **Inputs:** Based on whether the verifier has access to the auxiliary input, the protocol could be plaintext zero knowledge or auxiliary input zero knowledge. Our protocols allow for the verifier to have auxiliary input.

4 VERIFIABLE BINOMIAL MECHANISM

This section describes how to compute counting queries verifiably with differential privacy in both the single curator and client-server MPC models. We consider the trusted curator model to be a special instantiation of the general MPC model where the number of provers $K = 1$. In Section 4.1 we describe intuitions for our protocol, and in Section 4.2 we explain what is needed for verifiability in the MPC setting and tackle the additional challenges of verifying client inputs. We describe how prior efforts at verifying clients fall short of the security expectations of Definition 3.1. Finally, in Section 4.3, we describe a protocol that verifiably computes counting queries with DP.

Set $\mathcal{X} = \mathbb{Z}_q = \mathcal{Y}$, where $\mathbb{Z}_q$ is a prime order finite field of size q over the integers. Let $X = (x_1, \ldots, x_n)$ denote the client inputs and Q be the counting query $Q(X) = \sum_{i=1}^n x_i$. Let $[\![x_i]\!]_k$ denote the k'th additive secret[6] share of a client input x_i. Each client splits their input into K secret shares and distributes them across the provers. We will assume that $n \ll q$ and $\kappa = \lfloor \log_2 q \rfloor$ can be viewed as the security parameter. For $K \geq 1$ provers and 1 verifier, define the oracle functionality $\mathcal{M}_{\mathtt{Bin}}$ in the ideal world as follows:

(1) $\mathcal{M}_{\mathtt{Bin}}$ receives public privacy parameters ϵ and δ. It then computes n_b (number of coins for binomial noise) based on Lemma 2.7.

(2) Let $\Big([\![x_1]\!]_k, \ldots, [\![x_n]\!]_k\Big)$ denote the inputs on the k'th prover's input tape. Each prover Pv_k is expected to compute $X_k = \sum_{i=1}^n [\![x_i]\!]_k$ and sends to $\mathcal{M}_{\mathtt{Bin}}$ as its input X_k. A corrupted prover might send an arbitrary input.

(3) $\mathcal{M}_{\mathtt{Bin}}$ samples $\Delta_k \sim \mathtt{Binomial}(n_b, 1/2)$ independently for each input X_k it receives. It then computes

$$y = \textstyle\sum_{k=1}^K (X_k + \Delta_k) \tag{7}$$

(4) $\mathcal{M}_{\mathtt{Bin}}$ sends the tuple (y, Δ_k) as output to each prover Pv_k. On receiving its output, the Pv_k sends $\mathtt{CONTINUE}$ to $\mathcal{M}_{\mathtt{Bin}}$. Once $\mathcal{M}_{\mathtt{Bin}}$ receives the continue signal from prover Pv_k it moves on to deliver output to Pv_{k+1}.

(5) After all K provers have sent $\mathtt{CONTINUE}$, $\mathcal{M}_{\mathtt{Bin}}$ sends y as output to the verifier $\mathtt{Vfr}$. If a single prover fails to send the continue message and thereby exits the protocol early, the verifier and the remaining provers do not receive any output.

When $K = 1$, i.e., the trusted curator setting, the single prover receives n client inputs in plaintext, so $[\![x_i]\!]_k = x_i$ for all $i \in [n]$. This is equivalent to an adversary corrupting all K provers. Thus

[5] As $\mathcal{M}$ is a random function, the *joint distribution* of the view of the adversary and their output must be indistinguishable from the simulated transcript (and not just the view of the adversary). See [46] for more details.

[6] Although we describe our protocols with additive secret sharing, any linear secret sharing such as Shamir's secret sharing also applies to all our results.

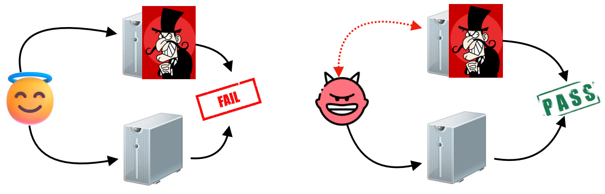
in the MPC setting with $K \geq 2$ servers, it is safe to assume at least one of them will follow the protocol. Our goal is to be able to come up with an interactive protocol $\Pi_{\texttt{Bin}}$, which allows us to compute $\mathcal{M}_{\texttt{Bin}}$ verifiably as per Definition 3.1. Notice that in the ideal model definition above, the oracle adds K independent copies of DP noise to the output, whereas Lemma 2.7 only calls for a single copy. This is because, as we allow up to $K-1$ provers to collude with a corrupted verifier, the corrupted provers could simply not add any noise to the output. Ben Or *et al.* 's completeness results [11] imply that K independent copies of noise are *necessary* to guarantee differential privacy unless the number of corruptions can be restricted to being strictly less than $\frac{K}{3}$, so each prover must independently generate enough noise to guarantee DP. Our protocols defined below are secure against computationally bounded provers and verifiers that may deviate arbitrarily from protocol specifications and have access to auxiliary inputs.

4.1 An Intuitive But Incomplete Protocol

Before describing the entire protocol in Section 4.3 and Figure 3, we provide the reader with some intuition as to why the protocol works for a single curator and verifier. *In this section, we make the unrealistic assumption that prover and verifier behave faithfully.* Assume all parties have joint oracle access to $\mathcal{O}_{\texttt{Morra}}$ (as described in Section 2.2) to jointly sample unbiased bits $(b_1, \ldots, b_{n_b})$. It is easy to see that using $(\sum_{i=1}^{n_b} b_i)$ as DP randomness results in the desired distribution defined in $\mathcal{M}_{\texttt{Bin}}$. However, the oracle output is known to both the verifier and prover; therefore, it cannot be directly used to guarantee differential privacy. As discussed earlier, this problem of proving that a prover faithfully sampled random bits without disclosing them lies at the heart of any verifiable DP protocol. Thus the protocol must combine public coins that satisfy verifiability requirements and private coins that ensure secrecy.

The protocol for verifiable DP counting proceeds in n_b identical and independent invocations (run in parallel). In copy i, the prover samples $v_i \in \{0, 1\}$, which it keeps private. Note that a prover could sample this bit using any arbitrary bias. As this is the provers' private coin, the verifier has no control over how the prover generates this information. After the prover has sampled their private bit, the prover and verifier make one call to $\mathcal{O}_{\texttt{Morra}}$ to get an unbiased coin denoted by b_i. Next, the prover locally computes $\hat{v}_i = b_i \oplus v_i$. Here $\oplus$ refers to the boolean XOR operation. It is easy to see that $\hat{v}_i$ has the same distribution as b_i, but its value is known only to the parties with access to v_i, i.e., the prover. After n_b rounds, the prover computes $Q(X)$ and $Z = \sum_{i=1}^{n_b} \hat{v}_i$ and outputs $Q(X) + Z$ where Z is used as DP randomness. By the assumption that the prover and verifier are faithful, Z is distributed according to the desired distribution stated in Theorem 2.7, and its value is only known to the prover. To make this protocol practical, we need to resolve a few issues.

(1) Although the above description requires a bitwise XOR operation to ensure the right distribution is used, we operate with arithmetic circuits in the actual protocol. Thus, the provers could sample arbitrary values $v^* \in \mathbb{Z}_q$ such that $v^* \notin \{0, 1\}$, and we need to fix how to express the XOR operation via arithmetic circuits.



(a) (b)

Figure 2: Two types of attacks that go undetected in Poplar. In (a) regardless of what the honest client sends, a corrupted server simply ignores the input and excludes the client from the protocol based on auxiliary information. In (b) a dishonest client colludes with the corrupted server by revealing secret values, so that an illegal input is included. In both cases, the honest server cannot distinguish between an honest run and a corrupted run of the protocol.

(2) Even if we could verify that the prover sampled a private bit correctly, we still need to verify that they faithfully performed the local operations discussed above.

Thus, if we could guarantee that each server performed its computations correctly and sampled a private value from the correct set, we would get the desired outcome of verifiable and DP counting queries.

4.2 Extending To Client-Server MPC-DP

To compute DP histograms verifiably in the client-server MPC-DP setting, we use the same computational model used for PRIO [25] and Poplar [15]. Prio is deployed at scale by Mozilla [7]. As discussed earlier, in this setting n clients secret share their inputs $x_i \in L$ amongst $K \geq 2$ provers, where $L \subseteq \mathcal{X}$ defines the language of legal inputs to the protocol. For computing M-bin histograms over n inputs, L is the set of all one-hot encoded vectors of size M. For the core problem of a single-dimensional counting query, $M = 1$ and $L = \{0, 1\}$. Since the inputs on the prover's tapes reveal no information about a client's input, for the protocol to be useful the provers must first verify in zero knowledge that $x_i \in L$ before using such inputs to compute aggregate statistics. This additional step of verifying a client is not required in the trusted curator model, as the prover decides what inputs should be included in the computation and can see them in plaintext.

Verifying Clients in MPC-DP.

Poplar and PRIO use efficient sketching techniques from [17] to validate a client's input in zero knowledge *without* relying on any public key cryptography. Thus, as long as at least one out of K provers does not reveal the inputs it received, even an unbounded adversary corrupting the remaining provers cannot ascertain any information about an honest client's input. While such a system protects an honest client's privacy from an unbounded adversary, it is not verifiable as per Definition 3.1. Specifically, for the techniques used in PRIO and Poplar, a single corrupted prover could tamper with its inputs and exclude an honest client from the protocol by

[7] https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/

forcing them to fail the verification test. Alternatively, a corrupt client could collude with a prover to include arbitrary inputs, jeopardising the correctness of the output. Figure 2[8] summarises these attacks on Poplar and PRIO[9]. By our definitions of verifiability, the protocol's output *must* be a function of the inputs provided by honest clients only. Thus the protocol described in Section 4.3 provides the following additional guarantees:

1. **Guaranteed Inclusion Of Honest Clients**: If a client submits shares of an input $x \in L$, then the final output of the protocol is guaranteed to use this input untampered. Thus an honest client is assured that, as long as a single prover follows the protocol specifications, no one learns any information about their private input and their input is correctly used to compute the final output.
2. **Guaranteed Exclusion Of Corrupt Clients**: A corrupted client, even one that has control over any proper subset of the K provers, cannot include an invalid input to the protocol. Thus if $x \notin L$, x is discarded by our protocol with overwhelming probability.

It is important to note that as we operate under stricter notions of privacy and correctness, our results require the use of public-key cryptography and security holds only against computationally bounded adversaries. Furthermore, we show in Section 5 that it is impossible to satisfy verifiable DP and provide information theoretic guarantees.

4.3 Main Protocol Description

The protocol $\Pi_{\mathtt{Bin}}$ described in Figure 3 provides a compact standalone description of the interaction between K provers and the verifier for computing $\mathcal{M}_{\mathtt{Bin}}$. We assume that both the provers and the verifier have access to oracles $O_{\mathtt{morra}}$ and O_{OR} as defined in Section 2.2. In the real world, $O_{\mathtt{Morra}}$ is replaced with $\Pi_{\mathtt{Morra}}$ (see Algorithm 1) and O_{OR} is replaced by Cramer *et al.* 's Σ-OR proof [26] (see Appendix B for an example implementation) which securely compute the oracle functionalities in the presence of adversaries that may deviate from protocol specifications. Thus, we define our protocol in the hybrid world, and by the sequential composition theorem[10] [37], the security properties of the protocol are preserved. Next, we describe the protocol in detail with line references to Figure 3:

Line 1: In the first step, the prover(s) and verifier agree upon the public parameters for the protocol. The public parameters include a description of $\mathcal{C}_{\mathsf{pp}} = \mathbb{G}_q, \mathcal{M}_{\mathsf{pp}} = \mathcal{X} = \mathcal{Y} = \mathbb{Z}_q, \mathcal{R}_{\mathsf{pp}} = \mathbb{Z}_q$ and a description of $\mathcal{M}_{\mathtt{bin}}$ as defined in equation (7). The group $\mathbb{G}_q$ satisfies the requirements of the homomorphic commitment scheme defined in Section 2.3 and we assume that the discrete log problem is hard to solve in $\mathbb{G}_q$.

Line 2: For each client $i \in [n]$, let $[\![x_i]\!]_k$ denote the k'th share of their input $x_i \in L$. Define $c_{i,k} = \mathrm{Com}\left([\![x_i]\!]_k, r_{i,k}\right)$ as the commitment to the k'th share of x_i. The client sends to each prover Pv_k the tuple $([\![x_i]\!]_k, r_{i,k})$ and broadcasts the commitments to each of the shares $\left(c_{i,1}, \ldots, c_{i,K}\right)$ to a public bulletin board that is observable to all parties.

Line 3-4: Similar to PRIO and Poplar, we use $L = \{0, 1\}$, and thus verifier and the client use the oracle O_{OR} to check if the client's input is indeed a commitment to a bit. For input x_i, the verifier (and provers) sends to O_{OR} the derived commitment $c_i = \prod_{k=1}^{K} c_{i,k}$ and the client sends the openings $\left(x_i, \sum_{k=1}^{K} r_{i,k}\right)$. The oracle responds with $O_{\mathsf{OR}}(c_i) = 1$ if $x_i \in \{0, 1\}$ and c_i is a commitment to x_i. In the real world, we replace O_{OR} with a Σ-OR protocol [11]. This step resolves the issues presented in Figure 2, as an honest client cannot be excluded nor can a corrupt client input be included. From here on, the protocol only uses inputs from validated clients.

Line 5: Pv_k samples $(v_{1,k}, \ldots, v_{n_b,k})$ where $v_{j,k} \in \{0, 1\}$ (private random bit) and sends to the verifier commitments to $v_{j,k}$ for $j \in [n_b]$. Let $c'_{j,k} = \mathrm{Com}(v_{j,k}, s_{j,k})$ denote the commitment to $v_{j,k}$ with randomness $s_{j,k}$. To enforce consistency in notation and improve readability, we always use c to denote commitments to client inputs and c' to denote commitments to the prover's private inputs. Similarly, we will always use r and s to denote the randomness used for client input and prover bit commitments, respectively.

Line 6-7: The verifier uses O_{OR} to check if the messages sent by the prover were indeed commitments to 0 or 1 (similar to verifying client inputs). This step is essential for the boolean to the arithmetic conversion, as the linearisation of the XOR operation is only valid for values $v \in \{0, 1\}$ (see completeness property of Theorem 4.1).

Line 8-9: If for any $i \in n_b$, $O_{\mathsf{OR}} = 0$, the verifier aborts the protocol and broadcasts that Pv_k cheated. Otherwise, once all commitments are verified, the prover and verifier jointly invoke $O_{\mathtt{morra}}$ to get n_b *public* unbiased bits $(b_{1,k}, \ldots, b_{n_b,k})$.

Line 10: For all $i \in [n_b]$, based on the value of $b_{j,k}$, the prover sets $\hat{v}_{j,k}$ and $\hat{s}_{j,k}$ as follows

$$\hat{v}_{j,k} = \begin{cases} 1 - v_{j,k} & \text{if } b_{j,k} = 1 \\ v_{j,k} & \text{otherwise.} \end{cases}$$

$$\hat{s}_{j,k} = \begin{cases} 1 - s_{j,k} & \text{if } b_{j,k} = 1 \\ s_{j,k} & \text{otherwise.} \end{cases}$$

As long as $v_{j,k} \in \{0, 1\}$, the above set of equations is equivalent to setting $\hat{v}_{j,k} = v_{j,k} \oplus b_{j,k}$. An important feature of this step is that, conditioned on $b_{j,k}$, the operations described above are linear. Line 11 describes why this is critical for correctness to hold.

Line 11: The prover sends (y_k, z_k) to the verifier:

$$y_k = \Big(\sum_{i=1}^{n} [\![x_i]\!]_k + \sum_{j=1}^{n_b} \hat{v}_{j,k} \Big) \tag{8}$$

[8] Content from J.J. at the English-language Wikipedia, licensed under CC BY-SA 3.0.

[9] Concretely, referring to notation from Appendix C of Poplar, in scenario (b), the dishonest client reveals the values κ and $[v]_0$ to the server. This allows the server to set $z_1 = -z_0, z_1^* = -z_0^*$ and $z_1^{**} = -z_0^{**}$, thereby admitting an illegal input into the protocol.

[10] Though we use sequential composition, both protocols $\Pi_{\mathtt{morra}}$ and $\Pi_{\mathtt{or}}$ can be parallelly composed.

[11] In the interactive setting, the verifier, the provers, and the client jointly sample public challenge by playing Morra. As long a single party is honest, the challenge is guaranteed to be selected uniformly at random. Alternatively, in the ROM model, the client sends to a public bulletin board a non-interactive Σ-proof using the Fiat-Shamir transform.

	Verifier(Vfr)		**Prover**(Pv_k)
1 :	$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$	Generate public parameters	$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$
2 :	$\left\{\left\{c_{i,k}\right\}_{k\in[K]}\right\}_{i\in[n]}$	Client inputs	$\left\{[\![x_i]\!]_k, r_{i,k}\right\}_{i\in[n]}, \left\{\left\{c_{i,k}\right\}_{k\in[K]}\right\}_{i\in[n]}$
3 :	$\forall i \in [n]$ Send $c_i = \prod_{k=1}^{K} c_{i,k}$	$\mathcal{O}_{\mathsf{OR}}$	$\forall i \in [n]$ Send $c_i = \prod_{k=1}^{K} c_{i,k}$
4 :	For any $i \in [n]$ if $\mathcal{O}_{\mathsf{OR}}(c_i) \neq 1$	Exclude $([\![x_i]\!]_k, r_{i,k})$ from the protocol	
5 :	$(c'_{1,k}, \ldots, c'_{n_b,k})$	$\xleftarrow{\quad c'_{j,k} = \mathrm{Com}\left(v_{j,k}, s_{v_{j,k}}\right) \quad}$	$\forall j \in [n_b]$ Samples and commits $v_{j,k} \in \{0,1\}$
6 :	$\forall j \in [n_b]$ Send $c'_{j,k}$	$\mathcal{O}_{\mathsf{OR}}$	$\forall j \in [n_b]$ Send openings$(v_{j,k}, s_{j,k})$
7 :	$\forall j \in [n_b]$ Check $\mathcal{O}_{\mathsf{OR}}(c'_{j,k}) = 1$		
8 :	$\forall j \in [n_b]$ Send empty string λ_j	$\mathcal{O}_{\mathsf{Morra}}$	$\forall j \in [n_b]$ Send empty string λ_j
9 :	Receive $(b_{1,k}, \ldots, b_{n_b,k})$	$\forall j \in [n_b]$ $b_{j,k} = \mathcal{O}_{\mathsf{Morra}}(\lambda_j)$	Receive $(b_{1,k}, \ldots, b_{n_b,k})$
10 :			$\forall j \in [n_b]$ Update $v_{j,k}, s_{j,k}$ to get $\hat{v}_{j,k}, \hat{s}_{j,k}$ based on $b_{j,k}$
11 :		$\xleftarrow{\quad (y_k, z_k) \quad}$	$y_k = \sum_{i=1}^{n} [\![x_i]\!]_k + \sum_{j=1}^{n_b} \hat{v}_{j,k}$ and $z_k = \left(\sum_{i=1}^{n} r_{i,k} + \sum_{j=1}^{n_b} \hat{s}_{j,k}\right)$
12 :	Compute $\hat{c}'_{j,k}$ using $b_{j,k}$ for all $j \in [b_{n_b}]$		
13 :	Check that $\left(\prod_{i=1}^{n} c_{i,k} \times \prod_{j=1}^{n_b} \hat{c}'_{j,k}\right) = \mathrm{Com}(y_k, z_k)$		

Figure 3: The figure above describes the interaction between a single prover and verifier in Π_{Bin}. In the single trusted curator model $K = 1$ we have $x_i = [\![x_i]\!]_k$ where the prover can see client inputs in plaintext. In the MPC setting, each prover Pv_k follows the exact same protocol on their respective inputs specified in Line 2. Thus at the end of the protocol, each prover Pv_k outputs the tuple y_k, z_k. A verifier aggregates the output from each prover to publish verifiable DP statistics.

$$z_k = \left(\sum_{i=1}^{n} r_{i,k} + \sum_{j=1}^{n_b} \hat{s}_{j,k}\right) \tag{9}$$

where (y_k, z_k) is the output for prover Pv_k.

Line 12: Using the common public randomness $\{b_{j,k}\}_{j\in[n_b]}$ generated by $\mathcal{O}_{\mathsf{morra}}$, the verifier updates their view of received commitments as follows:

$$\hat{c}'_{j,k} = \begin{cases} \mathrm{Com}(1,1) \times c'^{-1}_{j,k} & \text{if } b_{j,k} = 1 \\ c'_{j,k} & \text{otherwise.} \end{cases}$$

Note that Pv_k never opens $\hat{c}'_{j,k}$, and thus Vfr never sees $\hat{v}_{j,k}$ in plaintext. By the hiding property of commitments, an efficient verifier learns nothing about the prover's private values from these messages. However, as the update conditioned on $b_{j,k}$ is linear and $b_{j,k}$ is public, Vfr can still compute a commitment to $1 - v_{j,k}$ without ever knowing $v_{j,k}$. As a direct consequence, as discussed in the soundness claim, the prover cannot deviate from its prescribed linear operation, as the verifier can check it. As we will show later, this step guarantees correctness, soundness and security.

Line 13: Finally, the verifier checks

$$\prod_{i=1}^{n} c_{i,k} \times \prod_{j=1}^{n_b} \hat{c}'_{j,k} = \mathrm{Com}(y_k, z_k) \tag{10}$$

From these outputs, we can derive the desired result: we treat the y_k's as shares, and calculate $y = \sum_{k=1}^{K} y_k$ as the noisy sum. We next show that this protocol achieves our desired properties.

Theorem 4.1. *Let $X = (x_1, \ldots, x_n)$ be the client input. Let $\mathcal{M}_{\mathsf{Bin}}$ and $\mathcal{O} = (\mathcal{O}_{\mathsf{morra}}, \mathcal{O}_{\mathsf{OR}})$ be as defined above. Π_{Bin} is a verifiably differentially private argument with perfect completeness, negligible soundness and is computational zero knowledge.*

Proof.

Completeness: By the definition of $\mathcal{O}_{\mathsf{morra}}$, $(b_{1,k}, \ldots, b_{n_b,k})$ are all unbiased bits. As per Π_{Bin}, when $b_{j,k} = 1$, $\hat{v}_{j,k} = 1 - v_{j,k}$ and when $b_{j,k} = 0$, $\hat{v}_{j,k} = v_{j,k}$. We know that an honest prover is guaranteed to have sampled a private value $v_{j,k} \in \{0,1\}$ for all $j \in [n_b]$. Thus the case-wise arithmetic operation described above is equivalent to setting $\hat{v}_{j,k} = v_{j,k} \oplus b_{j,k}$. This implies that for each server $\hat{v}_{j,k} \xleftarrow{R} \{0,1\}$ and $\sum_{j=1}^{n_b} \hat{v}_{j,k} \sim \mathtt{Binomial}(n_b, 1/2)$. The output of each honest prover is thus $y_k = \mathtt{Binomial}(n_b, 1/2) + \sum_{i=1}^{n} [\![x_i]\!]_k$. By linearity of secret-sharing, $\sum_{k\in[K]} y_k = \mathcal{M}_{\mathsf{Bin}}(X, Q)$ where $\mathcal{M}_{\mathsf{Bin}}$ is defined in equation (7).

Soundess. Beyond exiting the protocol early (which is trivially detected), an adversary $\mathscr{A}$ controlling a collection of dishonest provers could force a prover to cheat by doing at least one of the following:

(1) (Cheat at Line 4): For any $j \in [n_b]$, $c'_{j,k}$ is not a commitment to a bit. As the verifier has access to oracle $\mathcal{O}_{\mathtt{OR}}$, it would detect this immediately. Thus we can be guaranteed that $c'_{j,k}$ are commitments to 1 or 0.
(2) (Cheat at Line 7): The prover could sample improper public randomness. However, this is impossible as the verifier and prover jointly use $\mathcal{O}_{\mathtt{Morra}}$ to generate randomness.
(3) (Cheat at Line 10): Output messages $(y'_k \neq y_k, z'_k \neq z_k)$. If the verifier check from (Line 12) fails then the verifier knows Pv^*_k cheated. If $\mathrm{Com}(y_k, z_k) = \prod_{i=1}^{n} c_{i,k} \times \prod_{j=1}^{n_b} \hat{c}'_{j,k} = \mathrm{Com}(y'_k, z'_k)$, then $\mathcal{A}$ has broken the binding property of the commitment scheme. As $\mathcal{A}$ has negligible success in winning the discrete log game, it has a negligible chance at breaking the commitment scheme.

These are the only places where the Pv^* sends a message to the $\mathtt{Vfr}$ and thus we have our result.

Zero Knowledge. To prove zero knowledge we need to explicitly define the commitment scheme we are using. We use Pedersen Commitments which are defined as follows

$$\mathrm{Com}(x, r) = g^x h^r \tag{11}$$

where $\mathcal{R}_{\mathsf{pp}} = \mathcal{M}_{\mathsf{pp}} = \mathbb{Z}_q$ and $C_{\mathsf{pp}} = \mathbb{G}_q$ an abelian group where the discrete log problem is hard. To enhance readability, we will prove security for $K = 2$ provers and one verifier, but the result trivially generalises to $K \geq 2$ provers. To avoid confusion between the MPC and single curator setting, we defer the simpler security proof for single curators to Appendix C. Without loss of generality, assume that the verifier $\mathtt{Vfr}^*$ and Pv_1 have been corrupted by a PPT adversary $\mathcal{A}$ and that Pv_2 is honest. Sim receives on its input tape the inputs for Pv_1 and $\mathtt{Vfr}^*$. The ideal oracle functionality $\mathcal{M}_{\mathtt{Bin}}$ is defined as before. Let Sim denote shorthand for $\mathsf{Sim}_{\mathtt{Vfr}^*,\mathsf{Pv}_1}$. We construct the simulator as follows:

(1) Sim receives the public messages $\left\{ \left\{ c_{i,k} \right\}_{k \in [K]} \right\}_{i \in [n]}$ and sets $c_i = \prod_{k=1}^{K} c_{i,k}$.
(2) Sim internally invokes Pv_1 to receive inputs X_1. If Pv_1 was honest then $X_1 = \sum_{i=1}^{n} [\![x_i]\!]_1$. Of course, we have no control over $\mathcal{A}$, and X_1 could be any arbitrary value. The definition of security requires that we prove security using the actual inputs used by the real-world adversary $\mathcal{A}$ and not the ones it was handed to at the start of the protocol.
(3) Sim invokes $\mathcal{M}_{\mathtt{bin}}$ with input X_1 and receives (y, Δ_1) as defined in equation (7). Note Sim never has access to the honest party's input X_2 nor the randomness Δ_2 used by Pv_2 in the real protocol. It must simulate the messages and output of the real protocol from just its input and the output it receives from the ideal model.
(4) Sim sets $y_1 = X_1 + \Delta_1$ and computes $y_2 = y - y_1$, which by the definition of $\mathcal{M}_{\mathtt{Bin}}$, is equal to $(X_2 + \Delta_2)$.
(5) Sim samples $z_2 \xleftarrow{R} \mathcal{R}_{\mathsf{pp}}$ and sets $c_2 = \mathrm{Com}(y_2, z_2)$.
(6) Sim samples $c'_{2,2}, \ldots, c'_{n_b,2}$ such that $c'_{j,2} = \mathrm{Com}(1, s_{j,2})$ where $s_{j,2} \xleftarrow{R} \mathcal{R}_{\mathsf{pp}}$. It sets $c'_{1,2} = g^1 a_2$ where $a_2 = c_2 \times \left(\prod_{j=2}^{n_b} \hat{c}'_{j,2} \right)^{-1} \times \left(\prod_{i=1}^{n} c_{i,2} \right)^{-1} \times g^{-1}$. Notice that Sim is actually unable to open $c'_{1,2}$ but is never required to do so, as opening a commitment to a private value violates DP. The only information $\mathcal{A}$ can check is if $c'_{1,2}$ is a commitment to a bit, which it is. Thus the simulator artificially constructs a set of commitments that align like the real-world protocol, without having the slightest idea what the randomness used by Pv_2 actually was. It is able to do so due to the hiding property of the commitment scheme.
(7) Sim sends over $\{c_{j,2}\}_{j \in [n_b]}$ to $\mathcal{A}$ pretending to be the honest prover (Line 4 of Figure 3).
(8) Sim pretends to be the prover and jointly invokes $\mathcal{O}_{\mathtt{Morra}}$ with $\mathcal{A}$ to sample n_b unbiased public bits $(b_{1,2}, \ldots, b_{n_b,2})$.
(9) Sim sends y_2 and z_2 to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

□

4.4 Public Verifiability and Randomness

Notice that the verifier does not contribute a private input to the protocol, and its messages contain no private information either. Furthermore, any party (the clients or the provers) may view the messages sent and received by the verifier. The verifier's role is primarily to generate unbiased public randomness (that is independent of the prover's messages), which is used to ensure soundness. It samples a challenge for the Σ-OR proof to verify that the provers' private values are well formed. Additionally, it participates in Morra, to generate unbiased public coins to enforce the provers DP noise is sampled from the correct distribution. In the computational complexity literature, such a verifier is called a public coin verifier [12]. If there was another way to sample unbiased and reliable randomness without the verifier, anyone accessing the message transcript could verify if soundness holds. Consider the Random Oracle Model (ROM), where the verifier's randomness generation (Morra) is replaced by applying a random oracle on the prover and client messages. Further, consider that all messages from the prover(s) are sent to a public bulletin board along with the client's input commitments and timestamps, with the slight modification that the prover sends commitments and non-interactive proofs of validity *before* the clients send messages to the board. The order matters as this prevents the prover from adaptively selecting private values based on the clients' messages, thereby biasing the output of applying a ROM on the board's contents. This way, the provers' messages are guaranteed to be independent of the honest client's messages. Now, any party (including the clients or even one that did not participate in the protocol) can verify that the randomness is correctly generated (using the oracle on the bulletin board messages) and then perform the checks assigned to the verifier to ensure soundness holds. There is no longer a need for parties to play Morra to generate reliable randomness. Thus, we do not need an explicit verifier. Such a protocol, where the correctness of the output can be verified by a non-participating entity, even when all participants responsible for computing the output are corrupted, is said to be publicly or universally verifiable (Definition 1 of [6]). Public verifiability is a critical property for protocols such as E-voting [40], where one cannot trust a single verifier or a small group of provers to help compute the output reliably. Of course, the ROM model is a theoretical construct. In the real world, we do not

[12] https://en.wikipedia.org/wiki/Interactive_proof_system

have a provable instantiation of a random oracle. Thus the protocol described above is not verifiable unless we assume at least one party (the verifier or one of the provers) is semi-honest. This semi-honest participant ensures that the public randomness is sampled reliably. So the question beckons, can we upgrade interactive proofs of differential privacy from verifiability to public verifiability in the plain model? Next we show that public verifiability (as defined Definition 1 of [6]) is impossible for interactive proofs for differential privacy if all participants are corrupted. Thus, the trust assumptions in our protocol above are the best we can hope for.

Unlike deterministic E-voting protocols, the outputs of a differentially private mechanism are, by definition, random. Furthermore, the output is a function of the client's inputs and the prover's private randomness. In end-to-end auditable voting [2, 40] or publicly verifiable MPC [6], the output of the protocol is a deterministic function of the client inputs only. Correctness is measured with respect to the output of an ideal functionality computing the desired function over these inputs. For DP mechanisms, the prover is responsible for providing a private but random input, which is used along with client inputs to compute a DP statistic. Thus in this setting, the prover has more agency to affect the output than the computing parties in an universally auditable MPC. The core problem for verifiable differential privacy lies in verifying that the final DP randomness comes from the correct distribution (an unbiased binomial distribution in our case) without learning anything about the prover's private random sample. Thus to verify a claim about a DP statistic, at the very least, we need a source for public and verifiable randomness. This is to say that the DP randomness must be computed as a joint function of the prover's private randomness and reliable public randomness to enforce both secrecy and verifiability. Without reliable public randomness, we cannot make a meaningful claim about the final output distribution. Thus a source of verifiable public randomness is necessary for verifiable DP. Such sources of public randomness are often called random beacons in the blockchain literature [13]. In the plain model (without a common random string (CRS) or a random oracle), we either need a trusted party to generate public randomness or require that the public randomness be computed using MPC among the participants. In the protocol above, we generate public randomness using Morra, one possible MPC instantiation of a random beacon (based on the classic commit and reveal approach). MPCs based on Verifiable Delay Functions (VDFs) can also generate public randomness with guaranteed output delivery [14]. Both Morra and VDFs require that at least one participant be semi-honest. In general, if all participants of the randomness-generating MPC are corrupted, then we cannot guarantee reliable public randomness. Thus in the plain model, we cannot guarantee public verifiability if all parties are corrupted, giving us the following corollary.

Corollary 4.2. *Provided that there is at least one honest participant or a reliable source of public randomness (random beacon), the transcript of* $\Pi_{\mathtt{Bin}}$ *can be efficiently verified by any party (even one that did not participate in the protocol). Absent this, it is impossible to provide universal verifiability in the plain model.*

5 SEPARATION UNDER VERIFIABLE DP

We show that information theoretic verifiable DP is impossible in the trusted curator model. To prove our result stated in Theorem 5.2, we rely on the impossibility of secure coin flipping by [39].

Theorem 5.1 (Impossibility Of Tossing A Fair Coin). *[39] Let* $(\mathtt{Pv}, \mathtt{Vfr})$ *be a coin tossing protocol and let* $B_\lambda = \mathbb{E}[\mathtt{out}(\mathtt{Pv}, \mathtt{Vfr})(1^\kappa)]$ *be the bias of the output of such a protocol. Assuming that one-way-functions do not exist, then for any* $g \in \mathtt{poly}(\kappa)$*, there exists a pair of efficient cheating strategies* $\mathtt{Pv}^*$ *and* $\mathtt{Vfr}^*$ *such that the following holds: for infinitely many* κ*'s, for each* $j \in \{0, 1\}$ *either* $\Pr[\mathtt{out}(\mathtt{Pv}^*, \mathtt{Vfr})(1^\kappa) = j]$ *or* $\Pr[\mathtt{out}(\mathtt{Pv}, \mathtt{Vfr}^*)(1^\kappa) = j]$ *is greater than* $\sqrt{B_\kappa^j} - \frac{1}{g(\kappa)}$*, where* $B_\kappa^1 = B_\lambda$ *and* $B_\kappa^0 = 1 - B_\lambda$*. In particular for* $B_\lambda = \frac{1}{2}$*, the corrupted party can bias the outcome by almost* $\frac{1}{\sqrt{2}} - \frac{1}{2}$.

The theorem above states that it is impossible for two unbounded parties to jointly sample an unbiased public coin. The result is stronger than the impossibility result by Cleve [24], which states that it is impossible to jointly flip an unbiased coin if we allow parties to exit early. The theorem above states that it is impossible even if we guarantee no party exists the protocol early.

Theorem 5.2 (Information Theoretic Verifiable DP is impossible). *Any constant round interactive protocols* Π *for an DP-mechanism* $\mathcal{M}_{\mathtt{Bin}}$ *that satisfies Verifiable-DP (Definition 3.1) cannot have unconditional soundness and statistical zero knowledge.*

Proof. Verifiable DP requires that a verifier be able to guarantee that the randomness generated by a prover remains unbiased, without the verifier ever seeing the randomness. Theorem 5.1, states that it is impossible for two unbounded parties to even jointly sample a *public* unbiased coin without assuming one way functions. Thus commitment schemes are both necessary and sufficient to jointly sample an unbiased public coin.

The task of jointly sampling unbiased *private* randomness is harder. If two parties could sample unbiased private randomness, then they could just use the same protocol to sample unbiased public randomness, by revealing the randomness. Thus, commitment schemes are a necessary condition for verifiable DP. Commitments cannot be both statistically binding and hiding, thus unbounded soundness and statistical zero knowledge is impossible. □

Connection With Open Problem.

Definition 5.3 (α-useful mechanism). Fix $\alpha \in [0, 1]$. Let $u : \mathcal{X}^n \times \mathcal{Y} \to \in \{0, 1\}$ be an efficiently computable deterministic function. A mechanism $\mathcal{M}$ is α-useful for a utility function u if for some $Q \in \mathcal{Q}$ and for all $X \in \mathcal{X}^n$

$$\Pr_{y \leftarrow \mathcal{M}(X,Q)} [u(X, y) = 1] \geq \alpha \tag{12}$$

In his survey on the complexity of DP, Vadhan [56] asks the following question. Given $X \in \mathcal{X}^n$ and a differentially private mechanism $\mathcal{M} : \mathcal{X}^n \times \mathcal{Q} \to \mathcal{Y}$, is there an efficient utility function u that is α-useful when $\mathcal{M}$ is IND-CDP but not when $\mathcal{M}$ is information-theoretically DP. Groce *et al.* [38] show that if the range of u is in $\mathcal{R}^n$ and the utility is measured in terms of the $\mathcal{L}_p$-norm, then statistical-DP and computational DP are equivalent. Thus for the separation to hold, the range of u must have a more

[13] https://a16zcrypto.com/content/article/public-randomness-and-randomness-beacons/.

complex structure, such as a graph, a circuit or a proof. Bun *et al.* corroborate this result by describing a utility function such that u is infeasible (not impossible) when $\mathcal{M}$ is statistical DP and efficient when $\mathcal{M}$ is computational DP [19]. Similar to our definition of verifiability, their utility function u is cryptographic and unnatural from a data analysis point of view. Specifically, given $y = \mathcal{M}(X, Q)$, Bun *et al.* define the utility as the answer to the question of whether y is a valid zap proof [33] of the statement "there exists a row in X that is a valid message signature pair". Meanwhile, we define our utility function as an interactive proof, that checks whether the real protocol output y, is indistinguishable from the output of an ideal run of $\mathcal{M}$. In Theorem 5.2, we show that verifiable DP is impossible in the presence of computationally unbounded adversaries. This provides a candidate for a separation between statistical DP and computational DP.

However, there are some key differences between our formulation of utility and how it was originally posed. For example, in Bun *et al.* , the utility function u is a deterministic non-interactive function that receives the output y and a dataset X of message-signature pairs. The task of evaluating utility is separate from the task of computing DP statistics. In verifiable DP, both the DP statistic and utility are computed simultaneously via a constant round interactive protocol. Furthermore, the number of rounds of the utility function is a function of the privacy parameter ϵ. Another point of difference is that, in verifiable DP, the verifier performs the dual role of evaluating the utility of the mechanism and generating randomness that prevents a curator from cheating (although it does not ever see this randomness). In Bun *et al.* , the verifier's task is just to verify the proof. They are not involved in generating the DP noise. Although we show that information theoretic verifiable DP is impossible, our definitions allow the adversary more agency. Thus the two settings are not directly comparable. We defer finding stronger connections between verifiable DP and finding a utility function that separates DP as per [56] to future work.

6 PERFORMANCE

This section quantifies the computational cost of $\Pi_{\texttt{Bin}}$, our protocol for computing verifiable DP counting queries. All results reported below were run on a *single* core of an Apple M1 Mac and the code to reproduce these results can be found at https://anonymous.4open.science/r/Verifiable-Differential-Privacy-C6E0/README.md.

In all our experiments, we instantiate the homomorphic commitment scheme using Pedersen Commitments (PC) [51] over the Ristretto curve[14]. A single commitment operation requires two multiplications and one addition and takes 156 μs. We instantiate $\mathcal{O}_{\texttt{Morra}}$ using $\Pi_{\texttt{Morra}}$ described in Section 2.2. We instantiate $\mathcal{O}_{\texttt{OR}}$ with the non-interactive Fiat-Shamir transform of the Σ-OR protocol described in Appendix B using SHA-3[15] as the random oracle. In the experiments discussed below, each client $i \in [n]$ sends commitments to its inputs and a non-interactive Σ-OR proof of their validity. Additionally, each client sends the prover(s) openings to its commitments as described in Line 2 of Figure 3.

Table 1: The table below benchmarks the latency of each stage of $\Pi_{\texttt{Bin}}$ for computing single dimension counting queries with parameters $n = 10^6, \epsilon = 0.095, \delta = 10^{-10}$. For a fixed value of δ, an $\epsilon = 0.095$ corresponds to $n_b = 262144$ private coins for the binomial mechanism.

C-Verifiy	Bit-commit	P-Verify	Morra	Agg	Check
169 sec	53 sec	45 sec	33 sec	79 ms	189 ms

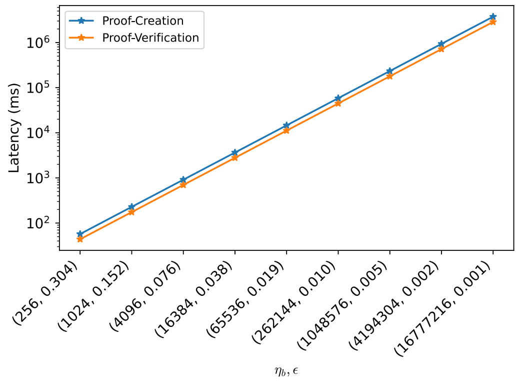


Figure 4: The figure above describes the latency of Σ-proof creation and verification as a function of the privacy parameters n_b and ϵ. For a fixed $\delta = 10^{-10}$, ϵ and n_b have one to one correspondence given by Lemma 2.7.

Table 1 describes the latency of different stages $\Pi_{\texttt{Bin}}$ with parameters $n = 10^6, \epsilon = 0.095, \delta = 10^{-10}$, in relation to Figure 3. Note that for the fixed value of $\delta = 10^{-10}$, $\epsilon = 0.095$ corresponds to $n_b = 262144$.

(1) The first column C-Verify describes the time it takes for the verifier to validate $n = 10^6$ client Σ-OR proofs sequentially (Lines 3-4).
(2) The second column Bit-commit describes the time it takes a single prover to sample n_b private bits and create n_b non-interactive Σ-OR proofs of their validity (Lines 5-6).
(3) The third column P-Verify, describes how long it takes the verifier to validate these proofs (Line 7).
(4) The fourth column describes the time it takes to play Morra, i.e., commit, open and aggregate n_b values in $\mathbb{Z}_q$ (Lines 9-10).
(5) The fifth column describes the time it takes to aggregate $n_b + n$ vales in $\mathbb{Z}_q$ (Line 11).
(6) Finally the last column describes the time it takes to check the provers outputs are correct (Lines 12-13).

We remark that numbers reported in the table result from running computations sequentially on a single core. As each round of $\Pi_{\texttt{Bin}}$ is independent of the other rounds, these computations could also be run in parallel. As our main bottleneck is working with the Σ-proof creation and verification, Figure 4 describes how proof creation and verification latency scales with the privacy parameter ϵ (or number of private coins n_b). Note that for high privacy settings (small values of ϵ), the prover(s) need to generate more private coins to ensure indistinguishability. Specifically, the number of coins (n_b) is proportional to $1/\epsilon^2$ (Lemma 2.7), and the time cost is then linear in n_b.

[14] https://doc.dalek.rs/curve25519_dalek/ristretto/struct.RistrettoPoint.html
[15] https://docs.rs/sha3/latest/sha3/

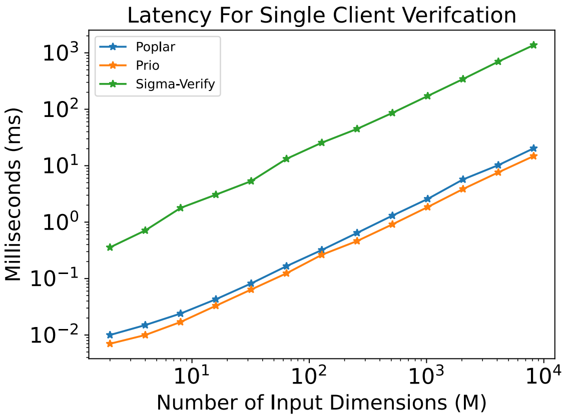


Figure 5: The figure above describes the drop in performance for using a Sigma protocol to verify that the client's commitment is wellformed. PRIO and Poplar use lightweight sketching protocols and general-purpose MPC to check in zero knowledge whether a client's input is a one-hot vector and do not need to assume one-way functions exist. But as described earlier, they are susceptible to collusion attacks.

Time cost for client verification (MPC case). Clients submit secret shares of their inputs in the MPC setting. Thus the servers must verify that the client inputs are valid. For M-dimensional DP-histogram estimation, the client inputs are restricted to one-hot encoded vectors of size M. As discussed in Section 4.2, the sketching techniques used in PRIO and Poplar allow servers to verify clients with information-theoretic security. Still, they are vulnerable to attacks by malicious servers. Our use of Σ-OR-protocols can defend against such attacks, but it comes at a higher computational cost due to its reliance on commitments (which assume one-way functions exist). Figure 5 benchmarks the increase in latency as a function of the number of dimensions (M) of client input. We remark that the numbers in Figure 5 are pessimistic as the Sigma-OR proof can be parallelised across the M dimensions (at the cost of communication complexity), whereas the sketching techniques cannot as they are based on the inner products.

7 RELATED WORK

Dwork *et al.* introduced DP and described the Laplace mechanism for outputting histograms in the trusted curator model [32]. Soon after, McSherry *et al.* proposed the exponential mechanism [49] (equivalently, report noisy max [29]), which lets us compute the (approximately) most frequent bucket in a histogram, also under pure differential privacy. Although these mechanisms give us pure differential privacy and optimal error rates $O(\frac{1}{\epsilon})$, implementing such a "central" model requires trusting that the curator to follow the protocol and not exploit the client data that it sees in plaintext.

Therefore, researchers studied local privacy (LDP) [43] using randomised response [57] to prevent any other party from seeing data in plaintext. Recently, Cheu, Smith and Ullman showed that the randomised response algorithm generalises all locally private protocols [23]. This generalisation highlights two unavoidable disadvantages of local differential privacy. The first is that the accuracy of the protocol for even the binary histogram is $O(\sqrt{n})$ compared to $O(1)$ in the central model. The second is that randomised response systems offer a much weaker definition of privacy than the usual cryptography standards such as semantic security. For example, if the client flips their original answer with probability $p = 0.1$, the curator sees their sensitive information in plain text 90% of the time. Further increasing p reduces the accuracy of the protocol dramatically. Consider the example from [25], where 1% of a million people answer "yes" to a survey about a sensitive topic. If we set $p = 0.49$, then one-third of the time the central analyser concludes that not a single member of the population answered "yes". Thus if we want to preserve utility, this definition of security is considerably weaker than the indistinguishability guarantees provided by protocols such as secret sharing.

Shuffle privacy [5, 22, 34] analyses local mechanisms under the lens of central privacy and bridges the accuracy gap between local and central models. Recent results [4, 36] prove that near central error guarantees are possible with distributed local transformations. Although this bypasses the accuracy issue of LDP, shuffle privacy assumes the existence of a secure shuffler, which is non-trivial to implement. In recent work, Bell *et al.* show that secure aggregation realises secure shuffling [8]. However, such protocols impose the impractical constraint of secure peer-to-peer communication between clients, and the curator is still a single source of failure. Despite the immense progress on differentially private histogram estimation, all known efficient implementations assume semi-honest participants and are a variant of either randomised response or the additive mechanism. It only takes a small fraction of clients to deviate from their prescribed protocol to destroy any utility of randomised response [23]. Additive mechanisms involve adding carefully curated randomness to the statistic before being released as output.

To ensure central DP error without a trusted curator, Dwork *et al.* proposed using standard MPC for computing DP statistics [31]. They proposed that each of the K servers would own a fraction of the entire dataset used for computation. As long as not more than $\lfloor \frac{K}{3} \rfloor$ of the servers are dishonest, it is possible to compute DP-histograms with optimal accuracy. However, the protocol is not publicly auditable and breaks down in presence of a dishonest majority of adversarial corruptions. McGregor *et al.* show a separation between DP obtained using a trusted curator and that obtained using MPC [48]. Specifically, they show that there exist computations (such as inner product or hamming distance) where mechanisms with $(1, 0)$-DP incur $\Omega(\sqrt{n})$ reconstruction error compared to $O(1)$ in presence of a trusted curator. To bridge this gap, Mironov *et al.* defined computational differential privacy, a relaxation of traditional DP [50]. They show that as long as semi-honest OT exists, it is possible to compute any computationally DP function with the same error rates as information theoretic DP in a trusted curator model. Histograms, unlike inner product and hamming distance, can be computed using MPC with the same error rates as trusted curator DP, under infomation theoretic DP. Thus recent work has focused on computing histograms using MPC.

Table 2: Summary of efforts MPC computation of aggregate DP statistics. The active security column describes if the protocols allowed participants to deviate arbitrarily. The Central DP column describes if the protocol output satisfies constant DP error independent of the number of clients participating in the protocol. The auditable property describes if the final output can be verified for correctness. Some interactive protocols leak additional information (such as prefix information about client input bits) beyond just the DP output. The leakage column describes if the prescribed protocols suffered from additional leakage.

Protocol	Active Security	Central DP	Auditable	Zero Leakage
Cryptographic RR [3]	✓			✓
Verifiable Randomization Mechanism [45]	✓		✓	✓
Securely Sampling Biased Coins [21]		✓		✓
MPC-DP heavy hitters[13]		✓		✓
PRIO [25]		✓		✓
Brave STAR [28]				
Sparse Histograms [8]		✓		
Crypt-ϵ [53]		✓		
Poplar [15]	✓	✓		
Our work	✓	✓	✓	✓

Bohler *et al.* use MPC to compute heavy hitters with semi-honest adversaries [13]. Researchers at Brave use oblivious pseudorandom functions (OPRF's) [41] and Shamir secret sharing [54] to compute k-anonymous histograms in the two server setting [28]. However, they do not include support for differential privacy. Researchers at Google use linear homomorphic encryption and OPRFs to compute differentially private sparse histograms in two-server models (2PC) [9], but require both the servers and clients to be semi-honest. Corrigan-Gibbs propose PRIO, a protocol in which a small number of servers receive arithmetic shares of client input to compute differentially private histograms [25]. PRIO uses shared non interactive proofs (SNIP's) to prevent clients from submitting illegal inputs but the protocol is only honest-verifier zero knowledge. Following the popularity of PRIO, Addanki *et al.* introduce PRIO+ to work over Boolean shares [1]. Boneh *et al.* use distributed point functions (DPFs) [18] to compute DP heavy-hitters in the two server model to propose a system called Poplar [15] that is zero knowledge even in presence of active adversaries. Roy *et al.* introduce *Crypt-ϵ*, a generic system to compute differentially private statisitcs using garbled circuits and linear homomorphic encryption [53]. The general purpose natue of *Crypt-ϵ* guarantees security only in the semi-honest threat model. Ambainis *et al.* proposed cryptographic randomised response [3] but are able to only guarantee local differential privacy. Table 2 summarises the assumptions under which the latest MPC protocols that have been used to compute DP statistics. As described earlier, existing work either assumes semi-honest adversaries or is not auditable. In 2021, the State Of Alabama sued the US deparment of commerce with regard to the errors caused due to random noise [42]. Differential Privacy by its defintion introduces a random noise blanket that tradesoff accuracy for privacy. This randomness is unavoidable if we wanted to protect individual privacy, but it also enables a corrupt aggregating server to disguise adversarial behaviour as randomness. In our paper, we first upgrade to security against active adversaries. Like existing literature we work in the dishonest majority model and further require the protocols to be publicly auditable. Our privacy constraints describe the most strict adversarial setting for practical deployment.

8 CONCLUDING REMARKS

We have introduced the notion of verifiable differential privacy to prevent malicious aggregators from using random noise as an attack vector. We have demonstrated the feasibility of this notion and showed that computational DP is necessary to achieve verifiability. A natural open question is to provide protocols for more complex DP mechanisms. Our protocol deliberately uses simple randomness (a Binomial distribution constructed from Bernoulli random variables), as making verifiable Laplace or Gaussian noise is far from clear. Similarly, approaches based on sampling from an appropriate distribution (the exponential mechanism) may be challenging since the distribution itself leaks information about the private data.

Acknowledgements. We thank the anonymous shepherd for guiding us through implementing our results on elliptic curves, improving the performance, and for helping us to clarify the limits of interactive proofs for differential privacy. This work is supported in part by the UKRI Prosperity Partnership Scheme (FAIR) under the EPSRC Grant EP/V056883/1, and the Alan Turing Institute

REFERENCES

[1] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. 2022. Prio+: Privacy preserving aggregate statistics via boolean shares. In Security and Cryptography for Networks. 516–539.

[2] Ben Adida. 2008. Helios: Web-based Open-Audit Voting.. In USENIX security symposium, Vol. 17. 335–348.

[3] Andris Ambainis, Markus Jakobsson, and Helger Lipmaa. 2004. Cryptographic randomized response techniques. In International Workshop on Public Key Cryptography. 425–438.

[4] Victor Balcer and Albert Cheu. 2020. Separating Local & Shuffled Differential Privacy via Histograms. arXiv:1911.06879 [cs] (2020). arXiv: 1911.06879.

[5] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. 2019. The privacy blanket of the shuffle model. In International Cryptology Conference. 638–667.

[6] Carsten Baum, Ivan Damgård, and Claudio Orlandi. 2014. Publicly auditable secure multi-party computation. In Security and Cryptography for Networks. 175–196.

[7] Carsten Baum, Alex J Malozemoff, Marc B Rosen, and Peter Scholl. 2021. Mac'n'Cheese : Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions. In International Cryptology Conference. 92–122.

[8] James Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure single-server aggregation with (poly) logarithmic overhead. In ACM CCS. 1253–1269.

[9] James Bell, Adria Gascon, Badih Ghazi, Ravi Kumar, Pasin Manurangsi, Mariana Raykova, and Phillipp Schoppmann. 2022. Distributed, Private, Sparse Histograms in the Two-Server Model. Cryptology ePrint Archive (2022).

[10] Robert M Bell and Yehuda Koren. 2007. Lessons from the netflix prize challenge. Acm Sigkdd Explorations Newsletter 9, 2 (2007), 75–79.

[11] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 2019. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali. 351–371.

[12] Manuel Blum. 1983. Coin flipping by telephone a protocol for solving impossible problems. ACM SIGACT News 15, 1 (1983), 23–27.

[13] Jonas Böhler and Florian Kerschbaum. 2021. Secure Multi-party Computation of Differentially Private Heavy Hitters. In ACM CCS. 2361–2377.

[14] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. 2018. Verifiable delay functions. In CRYPTO. 757–788.

[15] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight Techniques for Private Heavy Hitters. arXiv:2012.14884 [cs] (2021).

[16] danah boyd and Jayshree Sarathy. 2022. Differential Perspectives: Epistemic Disconnects Surrounding the US Census Bureau's Use of Differential Privacy. Harvard Data Science Review (Forthcoming) (2022).

[17] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions. In ACM CCS. 1292–1303.

[18] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2019. Secure computation with preprocessing via function secret sharing. In Theory of Cryptography Conference. 341–371.

[19] Mark Bun, Yi-Hsiu Chen, and Salil Vadhan. 2016. Separating computational and statistical differential privacy in the client-server model. In Theory of Cryptography Conference. 607–634.

[20] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In IEEE S&P. 315–334.

[21] Jeffrey Champion, Abhi Shelat, and Jonathan Ullman. 2019. Securely sampling biased coins with applications to differential privacy. In ACM CCS. 603–614.

[22] Albert Cheu. 2021. Differential privacy in the shuffle model: A survey of separations. arXiv preprint arXiv:2107.11839 (2021).

[23] Albert Cheu, Adam Smith, and Jonathan Ullman. 2021. Manipulation attacks in local differential privacy. In IEEE S&P. 883–900.

[24] Richard Cleve. 1986. Limits on the security of coin flips when half the processors are faulty. In ACM STOC. 364–369.

[25] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. arXiv:1703.06255 [cs] (2017).

[26] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 1994. Proofs of partial knowledge and simplified design of witness hiding protocols. In CRYPTO. 174–187.

[27] Ivan Damgård. 2000. Efficient concurrent zero-knowledge in the auxiliary string model. In the Theory and Applications of Cryptographic Techniques. 418–430.

[28] Alex Davidson, Peter Snyder, EB Quirk, Joseph Genereux, and Benjamin Livshits. 2021. STAR: Distributed Secret Sharing for Private Threshold Aggregation Reporting. arXiv preprint arXiv:2109.10074 (2021).

[29] Zeyu Ding, Daniel Kifer, Thomas Steinke, Yuxin Wang, Yingtai Xiao, Danfeng Zhang, et al. 2021. The permute-and-flip mechanism is identical to report-noisy-max with exponential noise. arXiv preprint arXiv:2105.07260 (2021).

[30] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. 2020. Line-point zero knowledge and its applications. Cryptology ePrint Archive (2020).

[31] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In the theory and applications of cryptographic techniques. 486–503.

[32] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In Theory of cryptography conference. 265–284.

[33] Cynthia Dwork and Moni Naor. 2000. Zaps and their applications. In Proceedings 41st Symposium on Foundations of Computer Science. 283–293.

[34] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. 2020. Amplification by Shuffling: From Local to Central Differential Privacy via Anonymity. arXiv:1811.12469 [cs, stat] (2020).

[35] Simson Garfinkel, John M Abowd, and Christian Martindale. 2019. Understanding database reconstruction attacks on public data. CACM 62, 3 (2019), 46–53.

[36] Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. 2020. On the Power of Multiple Anonymous Messages. arXiv:1908.11358 [cs, stat] (2020).

[37] Oded Goldreich. 2007. Foundations of cryptography. Vol. 1: Basic tools (digitally print. 1. paperback version ed.). Vol. 1. Cambridge Univ. Press, Cambridge.

[38] Adam Groce, Jonathan Katz, and Arkady Yerukhimovich. 2011. Limits of computational differential privacy in the client/server setting. In Theory of Cryptography Conference. 417–431.

[39] Iftach Haitner and Eran Omri. 2014. Coin flipping with constant bias implies one-way functions. SICOMP 43, 2 (2014), 389–409.

[40] Luke Harrison, Samiran Bag, Hang Luo, and Feng Hao. 2022. VERICONDOR: End-to-End Verifiable Condorcet Voting without Tallying Authorities. In ACM ASIACCS. 1113–1125.

[41] Stanisław Jarecki and Xiaomin Liu. 2009. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In Theory of Cryptography Conference. 577–594.

[42] Brennan Center For Justice. 2021. Alabama v. U.S. Dept of Commerce. https://www.brennancenter.org/our-work/court-cases/alabama-v-us-dept-commerce

[43] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? SICOMP 40, 3 (2011), 793–826.

[44] Shiva Prasad Kasiviswanathan, Mark Rudelson, and Adam Smith. 2013. The power of linear reconstruction attacks. In ACM-SIAM SODA. 1415–1433.

[45] Fumiyuki Kato, Yang Cao, and Masatoshi Yoshikawa. 2021. Preventing Manipulation Attack in Local Differential Privacy Using Verifiable Randomization Mechanism. In IFIP Conf. on Data and Applications Security and Privacy. 43–60.

[46] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. Tutorials on the Foundations of Cryptography (2017), 277–346.

[47] Ueli Maurer. 2009. Unifying zero-knowledge proofs of knowledge. In Cryptology in Africa. 272–286.

[48] Andrew McGregor, Ilya Mironov, Toniann Pitassi, Omer Reingold, Kunal Talwar, and Salil Vadhan. 2010. The limits of two-party differential privacy. In IEEE FOCS. 81–90.

[49] Frank McSherry and Kunal Talwar. 2007. Mechanism design via differential privacy. In 48th IEEE Symposium on Foundations of Computer Science (FOCS'07). 94–103.

[50] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. 2009. Computational differential privacy. In International Cryptology Conference. 126–142.

[51] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In CRYPTO. 129–140.

[52] Varun Raturi, Jinhyun Hong, David Philip McArthur, and Mark Livingston. 2021. The impact of privacy protection measures on the utility of crowdsourced cycling data. Journal of Transport Geography 92 (2021), 103020.

[53] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. 2020. Cryptϵ: Crypto-assisted differential privacy on untrusted servers. In ACM SIGMOD. 603–619.

[54] Adi Shamir. 1979. How to share a secret. CACM 22, 11 (1979), 612–613.

[55] Justin Thaler. 2020. Proofs, arguments, and zero-knowledge.

[56] Salil Vadhan. 2017. The complexity of differential privacy. In Tutorials on the Foundations of Cryptography. Springer, 347–450.

[57] Stanley L Warner. 1965. Randomized response: A survey technique for eliminating evasive answer bias. J. Amer. Statist. Assoc. 60, 309 (1965), 63–69.

[58] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In IEEE S&P. 1074–1091.

A FORMAL SECURITY DEFINITIONS

Definition A.1 (Discrete Log Assumption). For all PPT adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\Pr\left[x = x' : \begin{array}{c} (\mathbb{G}_q, g) \leftarrow \mathsf{Setup}(1^\kappa) \\ x \xleftarrow{R} \mathbb{Z}_q, h = g^x \\ x' \leftarrow \mathcal{A}(\mathsf{pp}, h) \end{array} \right] \leq \mu(\kappa)$$

Definition A.2. (Hiding Commitments) Let κ be the security parameter. A commitment scheme is said to be hiding for all PPT adversaries $\mathcal{A}$ the following quantity is negligible. The commitment is perfectly hiding if $\mu(\kappa) = 0$.

$$\Pr\left[b = b' : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa) \\ b \xleftarrow{R} \{0,1\}, r_{x_b} \xleftarrow{R} \mathsf{R}_{\mathsf{pp}} \\ (x_0, x_1) \in \mathcal{M}_{\mathsf{pp}}^2 \leftarrow \mathcal{A}(\mathsf{pp}) \\ c = \mathrm{Com}(x_b, r_{x_b}), b' = \mathcal{A}(\mathsf{pp}, c) \end{array} \right] \leq \mu(\kappa)$$

Definition A.3. (Binding Commitments) Let κ be the security parameter. A commitment scheme is said to be binding if, for all

	Verifier		Prover
1 :		Common Input $g, h, \mathbb{G}_q, q, c$	
2 :			$b, v_1, e_1 \overset{R}{\longleftarrow} \mathbb{Z}_q^2$; Set $d_0 = h^b$ and d_1 such that $d_1 \left(\frac{c}{g}\right)^{e_1} = h^{v_1}$
3 :	(d_0, d_1)	$\xleftarrow{\quad (d_0, d_1) \quad}$	(d_0, d_1)
4 :	$e \overset{R}{\longleftarrow} \mathbb{Z}_q$	$\xrightarrow{\quad e \quad}$	$e_0 = e - e_1 \mod q; v_0 = b + e_0 r$
5 :	Check $e_1 + e_0 = e$	$\xleftarrow{\quad (v_0, e_0, v_1, e_1) \quad}$	
6 :	Check $d_0 c^{e_0} = h^{v_0}$ and $d_1 c^{e_1} = g^{e_1} h^{v_1}$		

Figure 6: Proof for convincing Vfr that $c = gh^r$ is in $L_{\texttt{Bit}}$ without revealing that $x = 1$.

	Verifier		Prover
1 :		Common Input $g, h, \mathbb{G}_q, q, c$	
2 :			$b, v_0, e_0 \overset{R}{\longleftarrow} \mathbb{Z}_q^2$. Set $d_1 = h^b$ and d_0 such that $d_0 c^{e_0} = h^{v_0}$
3 :	(d_0, d_1)	$\xleftarrow{\quad (d_0, d_1) \quad}$	(d_0, d_1)
4 :	$e \overset{R}{\longleftarrow} \mathbb{Z}_q$	$\xrightarrow{\quad e \quad}$	$e_1 = e - e_0 \mod q; v_1 = b + e_1 r$
5 :	Check $e_1 + e_0 = e$	$\xleftarrow{\quad (v_0, e_0, v_1, e_1) \quad}$	
6 :	Check $d_0 c^{e_0} = h^{v_0}$ and $d_1 c^{e_1} = g^{e_1} h^{v_1}$		

Figure 7: Proofs for convincing Vfr that $c_x = h^{r_x}$without revealing the value x.

PPT adversaries $\mathcal{A}$, there exists a negligible function μ such that

$$\left| \Pr \left[(c_{x_0} = c_{x_1}) : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa) \\ x_0, r_{x_0}, x_1, r_{x_1} \leftarrow \mathcal{A}(\mathsf{pp}) \\ \text{s.t } x_0 \neq x_1 \end{array} \right] - \frac{1}{2} \right| \leq \mu(\kappa)$$

The commitment is perfectly binding if $\mu(\kappa) = 0$.

B OR PROTOCOL

Define as public parameters a cyclic prime order group $\mathbb{G}_q$ and generators g and h for $\mathbb{G}_q$. Let $\mathcal{M}_{\mathsf{pp}} = \mathcal{R}_{\mathsf{pp}} = \mathbb{Z}_q$. Pedersen Commitments defined below satisfy all properties described in Section 2.2.

$$\mathrm{Com}(x, r_x) = g^x h^{r_x} \tag{13}$$

For the sake of completeness, we describe the interactive disjunctive OR proof using Σ-protocols from [26]. Note that the Σ protocols are cheating verifier zero knowledge even without a random oracle. Maurer [47] shows that if the verifier's challenge space is polynomial sized, then the protocol can be shown to be zero knowledge. Damgard *et al.* show that by using Trapdoor commitments [27], one can preserve soundness and get zero knowledge but the protocol now has four messaging rounds instead of 3. Next, we describe the Σ-protocol that can be used to verify the OR condition.

Let $x \in \{0, 1\}$ and $c_x = \mathrm{Com}(x, r_x)$ for $r_x \overset{R}{\longleftarrow} \mathbb{Z}_q$ be the commitment to x. Given c_x, Π_{OR} is an interactive zero knowledge proof between a prover Pv and a verifier Vfr to show that $c_x \in L_{\texttt{Bit}}$. The security properties can be found in [26, 27, 55]. Figure 6 and Figure 7 succintly describe the OR protocol to prove that $c_x \in L_{\texttt{Bit}}$.

$$L_{\texttt{Bit}} = \{c_x : x \in \{0, 1\} \wedge c_x = \mathrm{Com}(x, r_x)\} \tag{14}$$

C DEFERRED SECURITY PROOFS

Single Curator Simulator Proof.

Theorem C.1. *Let* $\mathtt{Vfr}^*$ *denote the corrupted verifier. There exists a PPT Simulator* $\mathsf{Sim}_{(\mathtt{Vfr}^*)}$ *such that for all* $y = \mathcal{M}_{\mathtt{Bin}}(X, Q)$

$$\mathtt{View}\left[\Pi(\mathsf{Pv}, \mathtt{Vfr})\right] \overset{c}{\equiv} \mathsf{Sim}_{(\mathtt{Vfr}^*}(y, \vec{r}_v, z, \mathsf{pp})$$

where $z \in \{0, 1\}^{\mathsf{poly}(\kappa)}$ *and* $\vec{r}_v \in \{0, 1\}^{\mathsf{poly}(\kappa)}$ *represents auxiliary input and randomness available to all the corrupted parties.*

Proof. Denote the corrupted verifier as $\mathtt{Vfr}^*$. Sim receives on its input tape the inputs for $\mathtt{Vfr}^*$. The ideal oracle functionality $\mathcal{M}_{\mathtt{Bin}}$ is defined as before. Let Sim denote shorthand for $\mathsf{Sim}_{\mathtt{Vfr}^*}$. We construct the simulator as follows:

(1) Sim receives the public messages $\{c_i\}_{i \in [n]}$.
(2) Sim invokes $\mathcal{M}_{\mathtt{bin}}$ with the empty string λ and receives y as defined in equation (7).
(3) Sim samples $z \overset{R}{\longleftarrow} \mathcal{R}_{\mathsf{pp}}$ and sets $c = \mathrm{Com}(y, z)$.
(4) Sim samples $c'_2, \ldots, c'_{n_b}$ such that $c'_j = \mathrm{Com}(1, s_j)$ where $s_j \overset{R}{\longleftarrow} \mathcal{R}_{\mathsf{pp}}$. It sets $c'_1 = g^1 a$ where $a = c \times \left(\prod_{j=2}^{n_b} \hat{c}'_j\right)^{-1} \times \left(\prod_{i=1}^{n} c_i\right)^{-1} \times g^{-1}$.
(5) Sim sends over $\{c_j\}_{j \in [n_b]}$ to $\mathcal{A}$ pretending to be the honest prover (Line 4 of Figure 3).
(6) Sim pretends to be the prover and jointly invokes $\mathcal{O}_{\mathtt{Morra}}$ with $\mathcal{A}$ to sample n_b unbiased public bits $(b_1, \ldots, b_{n_b})$.
(7) Sim sends y and z to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

□